REMARKS

Claims 1-67 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 103(a) Rejections:

The Examiner rejected claims 1, 4-10, 19, 21-26, 36, 38-42, 48, 52-60, 62, 66 and 67 under 35 U.S.C. § 103(a) as being unpatentable over Huppenthal, et al. (U.S. Patent 6,339,819) (hereinafter "Huppenthal") in view of Hinds, et al. (U.S. Patent 6,542,916) (hereinafter "Hinds"), and further in view of Chen et al. (U.S. Patent 6,763,365) (hereinafter "Chen") claims 2, 3, 15-18, 27-29, 35 and 43-46 as being unpatentable over Huppenthal, Hinds and Chen, and further in view of Lasher et al. (U.S. Patent 4,863,247) (hereinafter "Lasher"), claims 11, 20, 30, 31, 37, 47 and 61 as being unpatentable over Huppenthal, Hinds and Chen, and further in view of Stribaek et al. (U.S. Patent 7,181,484) (hereinafter "Stribaek"), and claims 12-14, 32-34, 49-51 and 63-65 as being unpatentable over Huppenthal, Hinds and Chen, and further in view of Chen et al. (U.S. Patent 6,687,725) (hereinafter "Chen2"). Applicants respectfully traverse these rejections for at least the following reasons.

In the Response to Arguments section of the present Office Action, the Examiner submits that Applicants' Specification (on page 8, at paragraph [1056], lines 6-8; and on page 15, at paragraph [1076], lines 6-8]) discloses "the concept of the multiply-accumulate-chaining operation. The operation chaining concept discloses the chaining of operations in order to perform successive arithmetic operations in response to a single initiated instruction. The arithmetic operations can be multiplication, accumulation, carry-save, partial multiplication, and etc." The Examiner is clearly mischaracterizing the teachings of Applicants' Specification.

Paragraph [1056] of Applicants' Specification actually states, in its entirety, the following:

"Multi-word multiplications and additions may be computed using many word-sized multiply, add, and shift operations. As shown in FIG. 1, a 1024-bit integer X, for example, can be represented with sixteen 64-bit words X=(x15, . . . , x1, x0). That form is commonly referred to as multi-precision representation. The more efficiently a multi-word operation can be performed, the better the public-key performance will be. Adding capabilities to a general purpose processor to speed up multi-word operations is a key to accelerating public-key computations, and add-chaining and multiply-chaining are two of the capabilities needed."

This passage describes that adding add-chaining and multiply-chaining capabilities to a general purpose processor (such as by adding the new processor instructions of Applicants' claimed invention) will speed up <u>multi-word operations</u>, and therefore, can be used to accelerate public-key computations, which typically include such multi-word operations. This passage does not describe "multiply-accumulate-chaining," or "the chaining of operations in order to perform successive arithmetic operations in response to a single initiated instruction." In fact, it is not clear what the Examiner means by the phrase "the chaining of operations in order to perform successive arithmetic operations in response to a single initiated instruction." Applicants' claims describe <u>a relationship between two successive single arithmetic instructions</u>, not just a collection of arithmetic operations that are performed in response to a single initiated instruction.

Paragraph [1076] of Applicants' Specification actually states, in its entirety, the following:

The umulxck instruction is an efficient way to support public-key computations. Back-to-back scheduling of multi-word multiplications and accumulations is often difficult using today's instruction sets due to the multiplier latency. The umulxck instruction performs the multiply-accumulate-chaining operation which combines add-chaining and multiply-chaining in one operation and avoids the multiplier latency. Using the umulxck instruction, and referring again to FIG. 5, the calculation of a 64x1024 bit partial product (y0*X) and the accumulation with a previous partial product (s16:s0), can be accomplished in the following 20 instructions:

```
set register k=y0;
umulxck 0,0;                    // clear extended-carry register exc first
r0 = umulxck x0, s0;
 . . .
 . . .
. . .
r15 = umulxck x15, s15;
r16 = umulxck 0, s16;           // catch 64 carryout bits
r17 = umulxck 0, 0;             // catch last carryout bit
```

As described in Applicants' specification, <u>each single umulxck instruction</u> in this example performs both multiply and accumulate operations <u>and also implicitly allows for accumulating an additional partial result of a previous single umulxck instruction</u>, i.e. without requiring additional add operations or specifying an additional operand for the additional accumulate operation. Specifically, a umulxck instruction of the form shown above (rd = umulxck rs1, rs2 - where rs1 and rs2 are source operands and rd is a destination operand), performs (rs1*k + rs2 <u>+ a partial result of a previous single arithmetic instruction,</u> such as a previous umulxck instruction) <u>without explicitly specifying the additional operand</u> (the partial result of the previous single arithmetic instruction). The result register specified by operand rd receives the lower order bits of the result of this calculation, and the extended carry register (which is also not specified by any of the operands of the instruction) receives the upper order bits of the result of this calculation.

In the example referenced by the Examiner, the third single arithmetic instruction (r0 = umulxck x0, s0):

performs the calculations (x0*k + s0 + exc), where exc = 0 (because the second instruction above cleared exc)

stores the lower order bits of the result of this calculation in r0; and
stores the upper order bits of the result of this calculation in exc.

In this example, the fourth single arithmetic instruction (not shown, but implied to be (r1 = umulxck x1, s1):

performs the calculations (x1*k + s1 + exc), where exc now contains the upper order bits of the result of the calculation performed for the third instruction, as noted above;
stores the lower order bits of the result of this calculation in r0; and
stores the upper order bits of the result of this calculation in exc.

Similarly, each successive single umulxck instruction in this example performs a similar calculation on the explicitly specified source operands and also implicitly adds the contents of exc that were stored in exc by the previous umulxck instruction as the high order bits of the calculation made for that previous umulxck instruction.

As described in detail above, the Examiner's characterization of the teachings of Applicants' Specification is clearly incorrect. The cited portions of Applicants' Specification describe a specific processor instruction (umulxck). When a single instance of this specific processor instruction is executed by the processor, the processor performs specific multiply and accumulate calculations, including the implicit addition of a partial result of a previously executed single instance of the umulxck instruction. In other words, as noted above, execution of the processor instructions recited in Applicants' claims facilitates a feedback relationship between two successive single arithmetic instructions, not merely a collection of arithmetic operations that are performed in response to a single initiated instruction, or a general concept of operation chaining in performing a single arithmetic instruction. Because execution of these instructions facilitates this feedback relationship, the chaining of successive single instances of this umulxck instruction can be used to perform (and accelerate) the multi-word multiplies and accumulates that are common in cryptography applications.

In the Response to Arguments section of the present Office Action, the Examiner also submits, "The Huppenthal prior art discloses architecture for chaining a very large number of arithmetic operations (such as multiplication, accumulation, and etc) to form a single arithmetic instruction. The single arithmetic instruction is initiated and the entire sequence of chained instructions is performed with operand transfer controlled by the architecture via the usage of a chain port mechanism. The chain port mechanism supplies operands to each successive arithmetic operation in the sequence. The chain mechanism supplies operands without any support from the processor. The Hinds prior art discloses multiply-accumulate arithmetic operation sequences and its combination with Huppenthal discloses these arithmetic operational sequences completed as a single arithmetic instructions." Applicants assert that the Examiner is clearly mischaracterizing the teachings of both Huppenthal and Hinds in suggesting that the combination teaches Applicants' claimed invention.

Huppenthal is directed to a multiprocessor computer architecture incorporating a number of memory algorithmic processors ("MAP") in the memory subsystem or closely coupled to other processing elements (e.g., one or more <u>fixed instruction set</u> microprocessor-based processors) to enhance overall system processing speed. The memory algorithmic processor architecture is an assembly that contains field programmable gate arrays (FPGAs) functioning as the memory algorithmic processors. In other words, each memory algorithmic processor includes one or more FPGAs that are configurable to <u>perform various functions that are not performed by one of the instructions of the fixed instruction microprocessor-based processors in the system</u>. As described in the Summary section of Huppenthal, a MAP element can function autonomously from its host system (i.e. without processor intervention) <u>once its operands have been loaded</u>. MAP elements (which are not <u>processor instructions</u>, nor do they represent <u>circuitry to execute arithmetic instructions of a microprocessor instruction set</u>) can be chained together to forward operands from one to another when multiple MAP elements are working together to perform a very large function. As described in detail in Huppenthal, the MAP architecture, and FPGAs thereof, perform functions in response to <u>commands written to them using a write instruction</u> that specifies a command and operands in the data. This is clearly not analogous to the limitations of Applicants' claims, which require circuits for executing single arithmetic instructions of a processor instruction set. For example, Table 2 of Huppenthal (and the accompanying description) describe commands that are communicated to the MAP architecture by issuing a write instruction from processor 12. These commands are used to configure various elements of the MAP architecture (e.g., RMB, RUC, RECON, LDROM), to pass operands to the MAP architecture (e.g., WRTOP, LASTOP), and to initiate the performance of a function implemented in the MAP circuitry (e.g., START).

While performing a function using the MAP circuitry of Huppenthal may involve performing a series of calculations or other functions using one or more FPGAs, this teaches absolutely nothing about Applicants' claimed invention. Applicants' claims are not directed to the general concept of performing a collection of arithmetic operations to carry out a single arithmetic instruction, as the Examiner implies, or to the chaining of operations performed within a MAP architecture or FPGA in response to a command issued using one or more write instructions of a processor, but to specific arithmetic instructions in a processor's instruction set, individual instances of which implicitly add partial results of a previously executed single arithmetic instruction when executing a current single arithmetic instruction without explicitly specifying the partial result as an operand of the current single arithmetic instruction.

To establish a *prima facie* obviousness of a claimed invention, all claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974), MPEP 2143.03. The cited art does not teach or suggest all limitations of the currently pending claims, some distinctive limitations of which are set forth in more detail below.

Regarding claim 1, the cited art fails to teach or suggest *a first arithmetic circuit comprising a first plurality of arithmetic structures feeding back high order bits of a previously executed single arithmetic instruction of a processor instruction set in the public-key cryptography application, generated by the first arithmetic circuit, to a second* *arithmetic circuit comprising a second plurality of arithmetic structures* and *the second arithmetic circuit generating a first partial result of a currently executing single arithmetic instruction of the processor instruction set in the public-key cryptography application, wherein the currently executing single arithmetic instruction does not include an explicit source operand for specifying the high order bits, the first partial result representing the high order bits summed with low order bits of a result of a first number multiplied by a second number, the summing of the high order bits being performed during multiplication of the first number and the second number, the summing and at least a portion of the multiplication being performed in the second arithmetic circuit.*

The Examiner submits, "Huppenthal discloses a method implemented in a device and storing the first partial result; and using the stored first partial result in a subsequent computation in the public-key cryptography application, the method comprising: a previously executed single instruction <u>of a processor set</u> and <u>wherein the currently executing single arithmetic instructions does not include an explicit source operand for specifying the high order bits</u>" (emphasis Examiner's). **Applicants first assert that the Examiner has picked and chosen individual words and partial phrases taken from Applicants' claim in his remarks, completely ignoring the context in which they are used in the claim**. For example, the phrase "a method implemented in a device and storing the first partial result" has no meaning in the Examiner's remarks, as no partial result (or anything that could produce a result or partial result) is referenced by the Examiner. Similarly, the phrase "using the stored first partial result in a subsequent computation in the public-key cryptography application" has no meaning in the Examiner's remarks, since the Examiner has not identified a first computation with respect to the recited "subsequent computation." In addition, the phrase, "the method comprising: a previously executed single arithmetic instruction of a processor set" is incomprehensible, as it does not describe <u>a method step</u> at all. Finally, the phrase "wherein the currently executing single arithmetic instructions does not include an explicit source operand for specifying the high order bits" has no meaning in the

Examiner's remarks, since no such single arithmetic instructions or high order bits are referenced in his remarks.

The Examiner cites Huppenthal (in column 3, lines 1-7) as disclosing "number of MAP elements chained together to accomplish a single function or operation (implies a single arithmetic instruction)." **The Examiner's remarks regarding the implication of a single arithmetic instruction of a processor instruction set are completely unsupported by the reference itself.** In fact, as discussed above, the memory algorithmic processors taught by Huppenthal are explicitly disclosed as being separate from the fixed instruction set microprocessors in the system (such as processors 12_1 – 12_n). Instead, they act as co-processors implemented in FPGAs or another type of user array to perform algorithmic functions in response to commands issued to them using a <u>write instruction</u> of the fixed instruction set processor, which is clearly not an <u>arithmetic instruction</u> of the fixed instruction set processor.

The Examiner cites Huppenthal (in column 3, lines 18-25) as disclosing, "MAP elements can receive operands via chained port)" and (in column 18, line 66 – column 19, line 3) as disclosing, "output data from one MAP element to be sent directly to the user array of the next MAP element with no processor intervention via a chain port." Applicants again assert that the chaining of MAP elements to perform a large function coded in FPGAs and initiated by a write instruction teaches nothing about the limitations of Applicants' claim. For example, it does not teach circuitry that feeds back partial results of <u>one arithmetic instruction of a processor's instruction set</u> to circuitry executing <u>another arithmetic instruction of the processor's instruction set</u>. The chaining of MAP elements and passing of operands between MAP elements over a chain port when performing a <u>single large operation</u> implemented in one or more FPGAs, as described by Huppenthal, teach nothing about <u>feedback between arithmetic instructions of a processor's instruction set</u>.

The Examiner admits that Huppenthal does not specifically disclose a multiplying and summing sequence and relies on Hinds to teach such a sequence. Applicants again note that claim 1 does not merely require "a multiplying and summing sequence" but also requires feedback between two different arithmetic instructions. The Examiner submits that Hinds discloses a first arithmetic circuit comprising a first plurality of arithmetic structures feeding back high order bits of an executed arithmetic instruction in the public-key cryptography application, generated by the first arithmetic circuit, to a second arithmetic circuit comprising a second plurality of arithmetic structures and the second arithmetic circuit generating a first partial result of a currently executing arithmetic instruction in the public-key cryptography application, the first partial result representing the high order bits summed with low order bits of a result of a first number multiplied by a second number, the summing of the high order bits being performed during multiplication of the first number and the second number, the summing and at least a portion of the multiplication being performed in the second arithmetic circuit. **Again, the Examiner has selectively ignored some of the words explicitly recited in the limitations of Applicants' claim, changing the context and meaning of these limitations.** For example, claim 1 does not merely recite "feeding back high order bits of an executed single arithmetic instruction" but requires circuitry for "feeding back high order bits of a <u>previously</u> executed <u>single</u> arithmetic instruction <u>of a processor instruction set</u>." Similarly, claim 1 does not merely recite "the second arithmetic circuit generating a first partial result of a currently executing arithmetic instruction in the public-key cryptography application" but requires the second arithmetic circuit generating a first partial result of a currently executing <u>single</u> arithmetic instruction <u>of the processor instruction set</u> and that <u>the currently executing single arithmetic instruction does not include an explicit source operand for specifying the high order bits</u>.

The Examiner cites Hinds (in column 3, lines 5-17) as disclosing, "applying a multiply-accumulate operation to operands; multiplying operands and adding multiplication results" and (in column 8, lines 6-25) as disclosing, "chained multiply-accumulate operation; carry save adders and usage of partial multiplier (partial results: high order bits); output of results of partial multiplier is normalized and passed to carry save adders and final product adder)." Hinds is directed to an apparatus and method for implementing a floating point MAC instruction that takes three operands as inputs. The cited portions of Hinds describe the operation of an individual floating point MAC instruction, which includes both a multiplication and an addition operation. As described in Hinds, all three operands are explicitly specified for the MAC instruction; there is no additional operand that is implicitly added during execution of Hinds' MAC instruction. The Examiner's citation in column 8 refers to a prior art "chained" multiply-accumulate FPU, illustrated in FIG. 3B, which is operable to implement a single floating point multiply-accumulate operation (A+(B*C)) or floating-point multiply-subtract operation (-A+(B*C)) in response to issuance of a single floating-point instruction. This "chained" multiply-accumulate FPU passes results of a partial multiplier to a carry save adder and final product adder as part of executing a single floating point MAC instruction. This partial result is not implicitly fed back to a circuit implementing a subsequent arithmetic instruction, as in Applicants' claimed invention, nor is it fed back to a circuit implementing the current instruction from a circuit that generated a partial result of a previously executed instruction.

Applicants note that the Examiner has not provided any reason to combine the Huppenthal and Hinds references. As stated in *KSR Int'l Co. v. Teleflex Inc.*, No. 04-1350, slip. op. at 14 (U.S. Apr. 30, 2007), "rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal standard of obviousness." The Examiner must show that "there was an apparent reason to combine the known elements in the fashion claimed." *Id.* The Examiner's analysis "should be made explicit." *Id.* Since the Examiner has not articulated any reason to combine the references, the rejection is improper.

In addition, since the floating point MAC instruction of Hinds does not implicitly add a partial result from the previous execution of another floating point MAC (or a partial result from the previous execution of any other arithmetic instruction), as suggested by the Examiner and as required by Applicant' claim, the combination of Huppenthal and Hinds clearly would not result in Applicants' claimed invention. At most, it could result in a system in which the floating point MAC instruction of Hinds (which does not include such feedback) is implemented algorithmically in one or more FPGAs of the MAP architecture of Huppenthal, e.g., if such an instruction is not included in the instruction set of the fixed-instruction processors of Huppenthal.

The Examiner admits that Huppenthal-Hinds does not specifically disclose supporting a cryptography application and relies on Chen to disclose supporting a public-key cryptography application (citing Chen, column 6, lines 23-25, and "arithmetic operations to support acceleration of cryptographic functions"). The Examiner states, "It would have been obvious to one of ordinary skill in the art to modify Huppenthal-Hinds to support a cryptographic application as taught by Chen. One of ordinary skill in the art would have been motivated to employ the teachings of Chen to greatly improve the performance of cryptographic circuits. (Chen col. 5, lines 40-42)." The Examiner seems to imply that the reason to modify Huppenthal-Hinds to support cryptographic applications is to <u>improve the performance</u> of something that apparently <u>does not exist</u> in Huppenthal-Hinds (cryptographic circuits) for something that the Examiner admits that <u>Huppenthal-Hinds does not support</u> (i.e. cryptographic applications). Therefore, the Examiner has not provided a valid reason to combine the references. In addition, as described in detail above, the cited art does not teach all of the limitations of Applicants' claims, whether taken alone or in combination.

For at least the reasons stated above, Applicants assert that the Examiner has failed to establish a *prima facie* rejection of claim 1.

Independent claims 38 and 66 include limitations similar to those recited in claim 1 and discussed above, and were rejected for similar reasons. Therefore, the arguments presented above apply with equal force to these claims, as well.

Independent claim 21 includes limitations similar to those recited in claim 1 and discussed above, and was rejected for reasons similar to those discussed above regarding claim 1. Therefore, Applicants traverse the rejection of this claim for at least the reasons presented above regarding limitations in this claim that are similar to those in claim 1.

In addition, independent claim 21 recites *supplying a third number to the second arithmetic circuit* and *the first partial result being a representation of the high order bits summed with low order bits of a result of a first number multiplied by a second number and with the third number*. The Examiner submits that Hinds discloses these limitations using the same citations and reasoning as those included in remarks directed to claim 1. However, as discussed in detail above, Hinds teaches a floating point MAC instruction of the form (A+(B*C)), for which operands A, B, and C are explicitly specified. Hinds does not disclose a MAC instruction that feeds back partial results of a previously executed instruction, <u>the first partial result being a representation of the high order bits summed with low order bits of a result of a first number multiplied by a second number and with the third number</u>, as required by claim 21.

For at least the reasons above, the rejection of claim 21 is unsupported by the cited art and the removal thereof is respectfully requested.

Claims 53 and 67 include limitations similar to those recited in claims 1 and 21 and discussed above, and were rejected for the same reasons as claims 1 and 21. Therefore, the arguments presented above apply with equal force to these claims, as well.

Applicants assert that numerous ones of the dependent claims recite further distinctions over the cited art. Applicants traverse the rejection of these claims for at least the reasons given above in regard to the claims from which they depend. However, since the rejections have been shown to be unsupported for the independent claims, a discussion of the dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicants submit the application is in condition for allowance, and an early notice to that effect is requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/6000-31500/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant(s)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: May 4, 2010